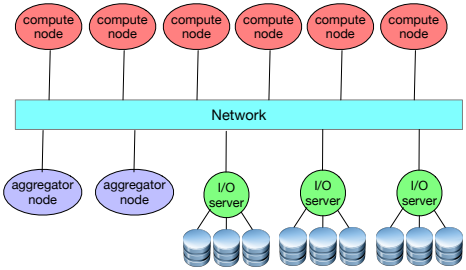
CS420 – Lecture 19

Marc Snir

Fall 2018

MPI-IO

- Compute nodes collectively write small chunks of a large shared file;
- Chunks that go to the same I/O server are aggregated into a large write operation by an aggregation node and sent to the I/O server.
  - Communication from compute nodes to aggregation noes uses MPI
  - Communication from aggregation nodes to server nodes uses IP
- Large number of small chunks are "transposed" into a small number of large chunks



Parallel file system: Lustre, GPFS

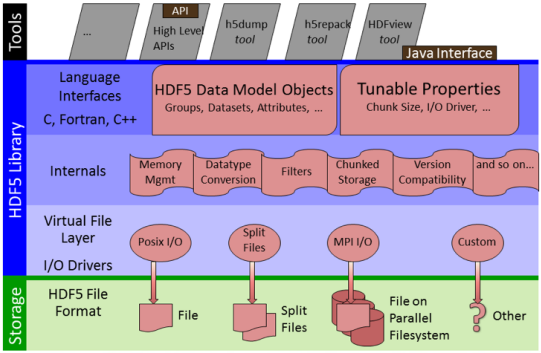
10,000's disks are aggregated into one file system

- Files are stripped across many disks (user may control stripping)
- Parity or error correcting codes ensure that the failure of one (or two) disks cause no loss of data
- Caching in memory hides disk latency

MPI-IO can be used for efficient access to files in the PFS

- Read = Get, Write = Put
- Collective read/write similar to all-to-all

HDF5

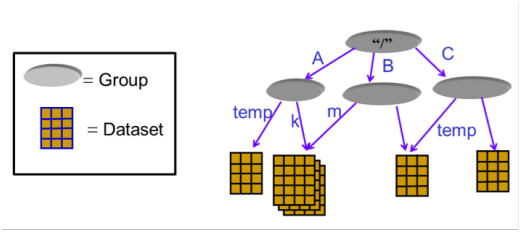


- Has its own data model: HDF5 files can be only accessed by HDF5
  - But there is a "dump" facility
- Is implemented atop
  - Posix I/O (one or multiple files),
  - GPFS/Lustre parallel file systems
- Has bindings for C/C++/Fortran/Python

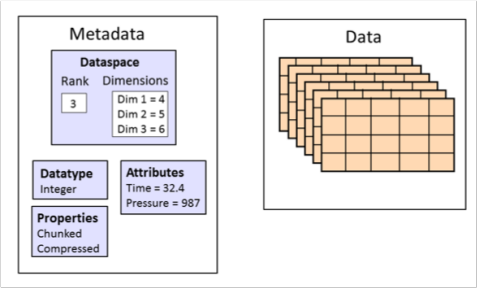
Will describe first (sequential) HDF5, next (parallel) PHDF5

Basic objects:

- **Group:** similar to directory
- **Dataset:** multidimensional array
- **Datatype:** describes type of dataset entries (e.g., 32 bit float)
- **Dataspace:** describes layout of dataset (number of dimensions and their size) or layout of subset of a dataset accessed by a read/write operation



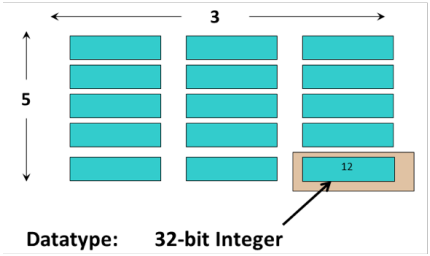
Dataset

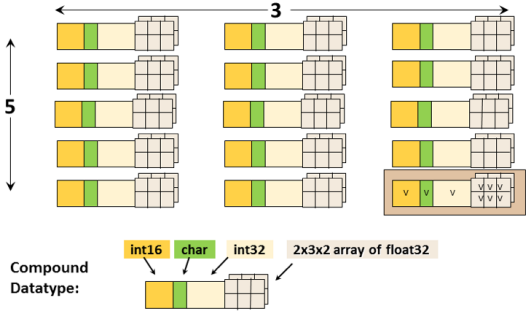


- Contains metadata and data
- Data is organized in a multidimension
- Metadata includes numeric attributes,

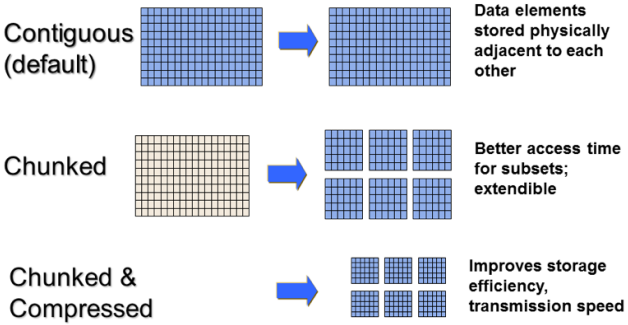
Datatype

Describes the type of each entry in the array (similar to MPI datatypes)



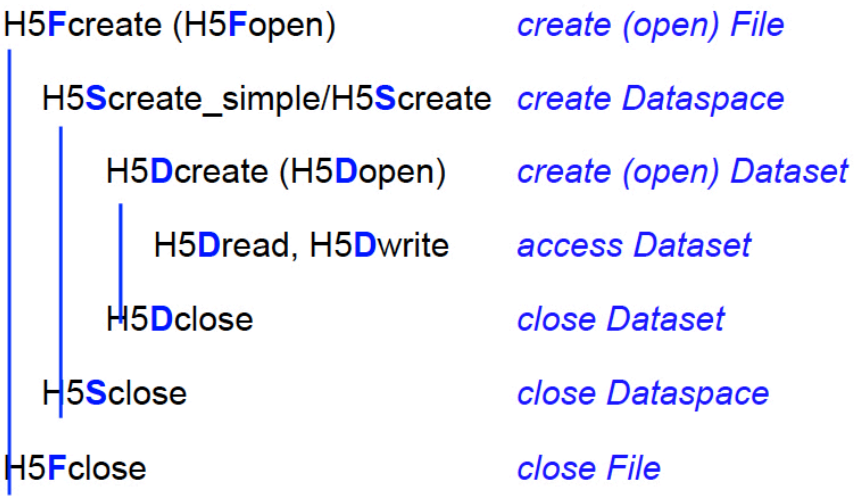


Storage properties



- Chunked = set of blocks
  - Can grow in each dimension
- Chunked & compressed: compress each block

HDF5 Use


H5Fcreate (H5Fopen)
create (open) File
H5Screate_simple/H5Screate
create Dataspace
H5Dcreate (H5Dopen)
create (open) Dataset
H5Dread, H5Dwrite
access Dataset
H5Dclose
close Dataset
H5Sclose
close Dataspace
H5Fclose
close File

File operations

```
hid_t file_id;
herr_t status;
file_id = H5Fcreate(''dset.h5'', H5F_AA_TRUNC,  H5P_DEFAULT, H5P_DEFAULT);
status = H5Fclose(file_id);
```

- All HDF5 object handles have type `hid_t`
- `herr_t` is the type of a status (errror message)
- All file (group) operations start with H5F

Create file (group)

```
H5Fcreate(name, flag, fcpl_id, fapl_id)
```

flag: access mode (`HF5_AA_TRUNC`: create new file or overwrite existing file – same as "w" in Posix)

fcpl_id: file creation property list – specifies the file metadata

fapl_id: file access property list – specifies the access method; could specify a communicator, for a parallel program

Dataset operations

```
/* create dataspace */
dims[0] = 4;
dims[1] = 6;
dataspace_id =  H5Screate_simple(2,dims,NULL);
/* create dataset */
dataset_id = H5Dcreate (file_id, ''/dset'', H5T_STD_I32BE, dataspace_id,
H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

status = H5Dclose(dataset_id);
status = H5Sclose(dataspace_id);
```

H5S – Dataspace operation

```
dataspace_id = H5Screate_simple(2,dims,NULL);
H5Screate_simple( rank, current_dims, maximum_dims)
```

rank: number of dimensions

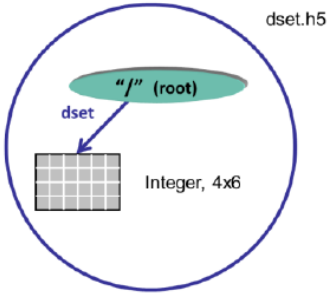
current_dims: size in each dimension

maximum_dims : max size in each dimension

- If NULL then max size = current size
- An entry in `maximum_dims` can have value `H5S_UNLIMITED`

H5D – Dataset operation

```
dataset_id = H5Dcreate (file_id, "/dset", H5T_STD_I32BE, dataspace_id,
H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
```



- Created within file `dset.h5` a dataset at location "/dset"
- The datatset is a 4×6 array of 32 bit integers (big endian)

`H5Dcreate(file_id, loc_id, dtype_id, space_id, lcpl_id, dcpl_id, dapl_id )`

- file_id: file (group) containing dataset
- loc_id: location identifier (file, or group within file)
- dtype_id: datatype of elements
- space_id: dataspace
- lcpl_id: link creation property list
- dcpl_id: dataset creation property list
- dapl_id: dataset access property list

Read and write

Copies an array into the dataset or vice-versa

```
int dset_data[4][6];

status = H5Dwrite (dataset_id, H5T_NATIVE_INT, H5S_ALL,
H5S_ALL, H5P_DEFAULT, dset_data);

status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL,
H5S_ALL, H5P_DEFAULT, dset_data);
```

```
H5Dwrite(dataset_id, mem_type_id, mem_space_id,
      file_space_id, xfer_plist_id, buf)
```

mem_type_id: memory datatype; if different than dataspace datatype, then conversion occurs. (`H5T_NATIVE_INT` – native integer; could be 64 bits)

mem_space_id: dataspace specifying what part of the in memory array is accessed (`H5S_ALL` if the entire array is accessed)

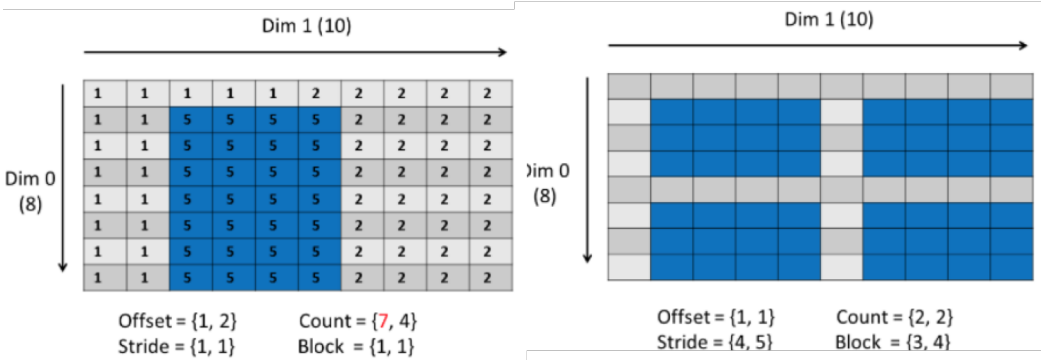
file_space_id: dataspace specifying what part of the dataset is accessed (`H5S_ALL` if the entire array is accessed)

xfer_plist: data transfer property list; in particular it determines if accesses are individual or collective.

buf: memory buffer

Dataspace can specify subarray

Similar to vector datatype in MPI



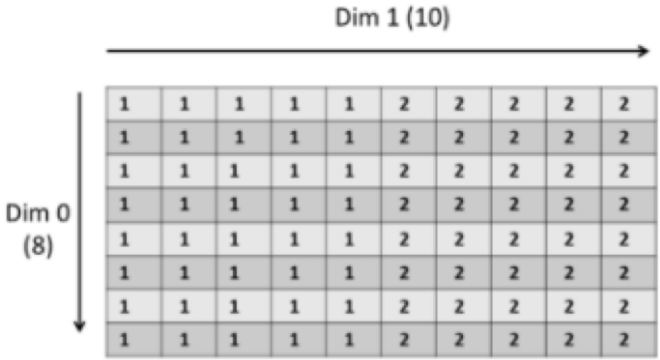
```
int main () {

int data[8][10]; /* data to write */
int i,j;
for(i=0;i<8; i++)
for(j=0; j<10; j++)
data[i][j] = j>4? 1:2;
```

Write entire dataspace

```
hsize_t dims[2]; /* hsize_t is integer type used for dataspace dimensions */
hid_t file, dataspace, memspace, dataset;
herr_t status;
file = H5Fcreate (''file.h5'', H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
dims[0] = 8;
dims[1] = 10;
dataspace = H5Screate_simple (2, dims, NULL);
dataset = H5Dcreate2(file, ''IntArray'', H5T_STD_I32BE, dataspace,
H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
status = H5Dwrite (dataset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
H5P_DEFAULT, data);
status = H5Sclose (dataspace);
status = H5Dclose (dataset);
status = H5Fclose (file);
```

Dataspace values



Contents of Original Dataset Created

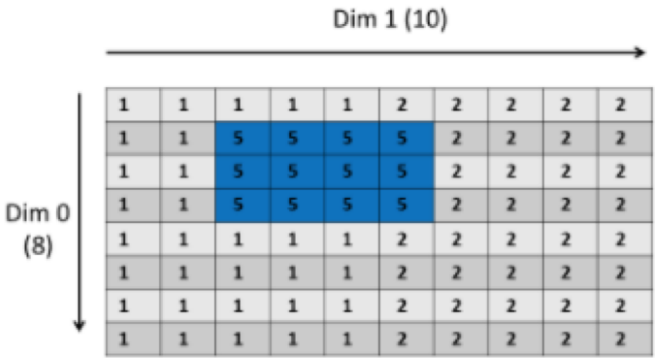
```
int sdata[3][4]
for (i=0; i<3; i++)
for (j=0;j<4; j++)
sdata[i][j]=5;

file = H5Fopen (''file.h5'', H5F_ACC_RDWR, H5P_DEFAULT);
dataset = H5Dopen (file, ''Intarray'', H5P_DEFAULT);

hsize_t offset[2] = {1,2};
hsize_t count[2] ={3,4};
hsize_t stride[2] = {1,1};
hsize_t block[2] = {1,1};
```

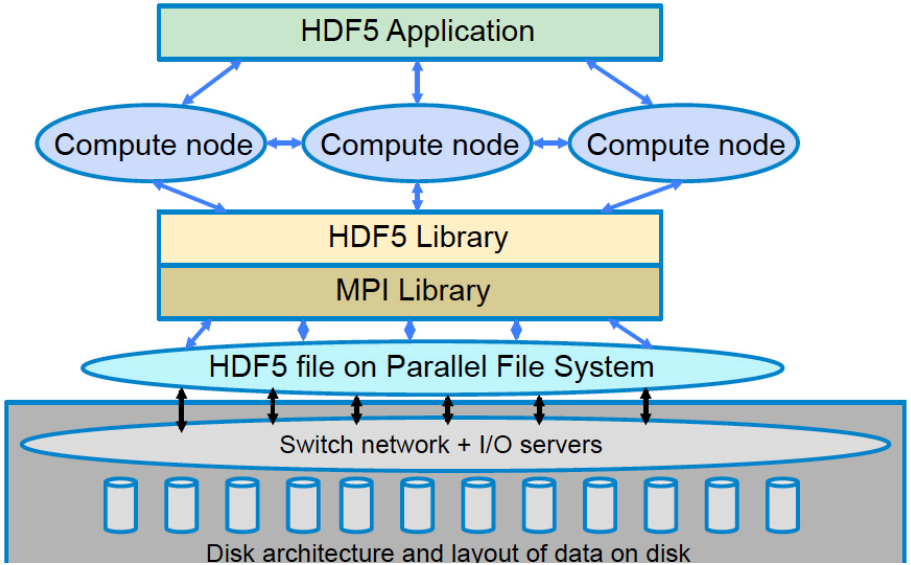
```
memspace = H5Screate_simple (2, count, NULL);
dataspace = H5Dget_space (dataset);
status = H5Sselect_hyperslab (dataspace, H5S_SELECT_SET,
offset, stride, count, block);
status = H5Dwrite (dataset, H5T_NATIVE_INT, memspace,
dataspace, H5P_DEFAULT, sdata);
```

Dataspace values



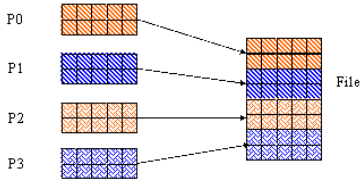
Contents of Dataset after 3 x 4 Subset Written

PHDF5


HDF5 Application
Compute node
Compute node
Compute node
HDF5 Library
MPI Library
HDF5 file on Parallel File System
Switch network + I/O servers
Disk architecture and layout of data on disk

- All metadata operations (on files, dataspaces, etc.) are collective
- Read/Write operations are either collective or individual
- Use same calls as in HDF5, with suitable property lists

Example: gather or scatter



Input/output slices of Jacobi matrix, or chunks of a list to be sorted

```
/*
*  This example writes data to the HDF5 file by rows.
*  Number of processes is assumed to divide number of rows
*/

#include "hdf5.h"
#include "stdlib.h"

#define H5FILE_NAME     "SDS_row.h5"
#define DATASETNAME   "IntArray"
#define NX      8                       /* dataset dimensions */
#define NY      5
#define RANK    2

int main (int argc, char **argv)
{
```

```
/*
* HDF5 APIs definitions
*/
hid_t       file_id, dset_id;         /* file and dataset identifiers */
hid_t       filespace, memspace;      /* file and memory dataspace identifiers
*/
hsize_t     dimsf[2];                 /* dataset dimensions */
int         *data;                    /* pointer to data buffer to write */
hsize_t count[2];           /* hyperslab selection parameters */
hsize_t offset[2];
hid_t plist_id;                 /* property list identifier */
int         i;
herr_t  status;
```

```
/*
* MPI variables
*/
int mpi_size, mpi_rank;
MPI_Comm comm  = MPI_COMM_WORLD;
MPI_Info info  = MPI_INFO_NULL;

/*
* Initialize MPI
*/
MPI_Init(&argc, &argv);
MPI_Comm_size(comm, &mpi_size);
MPI_Comm_rank(comm, &mpi_rank);

/*
* Set up file access property list with parallel I/O access
*/
plist_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(plist_id, comm, info);
```

```
plist_id = H5Pcreate(H5P_FILE_ACCESS);
```

```
hid_t H5Pcreate(hid_t cls_id)
```

Create property list as an instance of the specified property list class (in our case, file access property list)

```
H5Pset_fapl_mpio(plist_id, comm, info);
```

```
H5Pset_fapl_mpio(hid_t fapl_id, MPI_Comm comm, MPI_Info info)
```

Stores the communicator to be used for parallel I/O and additional Info into the file access property list.

`H5P` prefixes functions that manipulate property lists

```
/*
* Create a new file collectively and release property list identifier.
*/
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
H5Pclose(plist_id);


/*
* Create the dataspace for the dataset.
*/
dimsf[0] = NX;
dimsf[1] = NY;
filespace = H5Screate_simple(RANK, dimsf, NULL);
```

```
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
```

```
hid_t H5Fcreate( const char *name, unsigned flags, hid_t fcpl_id, hid_t
fapl_id )
```

Creates a file with specified name and property lists for creation and access, and specify whether to overwrite existing file

```
filespace = H5Screate_simple(RANK, dimsf, NULL);
```

```
hid_t H5Screate_simple( int rank, const hsize_t * current_dims, const
hsize_t * maximum_dims )
```

Creates simple dataspace (array) of specified rank, initial dimensions and maximum dimensions

```
/*
* Create the dataset with default properties and close filespace.
*/
dset_id = H5Dcreate(file_id, DATASETNAME, H5T_NATIVE_INT, filespace,
H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
H5Sclose(filespace);

/*
* Each process defines dataset in memory and writes it to the hyperslab
* in the file.
*/
count[0] = dimsf[0]/mpi_size;
count[1] = dimsf[1];
offset[0] = mpi_rank * count[0];
offset[1] = 0;
memspace = H5Screate_simple(RANK, count, NULL);
```

```
dset_id = H5Dcreate(file_id, DATASETNAME, H5T_NATIVE_INT, filespace,
H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
```

```
hid_t H5Dcreate( hid_t loc_id, const char *name, hid_t dtype_id, hid_t
space_id, hid_t lcpl_id, hid_t dcpl_id, hid_t dapl_id )
```

Creates dataset of specified name, within specified file (or group), with structure defined by dataspace and datatype, with specified creation, dataset creation and dataset access property lists.

```
/*
* Select hyperslab in the file.
*/
filespace = H5Dget_space(dset_id);
H5Sselect_hyperslab(filespace, H5S_SELECT_SET, offset, NULL, count, NULL);

/*
* Initialize data buffer
*/
data = (int *) malloc(sizeof(int)*count[0]*count[1]);
for (i=0; i < count[0]*count[1]; i++) {
data[i] = mpi_rank + 10;
}
```

```
filespace = H5Dget_space(dset_id);
hid_t H5Dget_space(hid_t dataset_id )
```

Get a handle to (a copy of) the dataspace of the specified dataset

```
H5Sselect_hyperslab(filespace, H5S_SELECT_SET, offset, NULL, count, NULL)
herr_t H5Sselect_hyperslab(hid_t space_id, H5S_seloper_t op, const hsize_t
*start, const hsize_t *stride, const hsize_t *count, const hsize_t *block )
```

Modifies specified dataspace using the specified operation(replace, add, OR, AND, ...), using start, stride, count and block arguments for the subarray

```
/*
* Create property list for collective dataset write.
*/
plist_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

status = H5Dwrite(dset_id, H5T_NATIVE_INT, memspace, filespace,
plist_id, data);
free(data);
```

```
plist_id = H5Pcreate(H5P_DATASET_XFER);
```

Create a property list for dataset read/write

```
H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);
herr_t H5Pset_dxpl_mpio( hid_t dxpl_id, H5FD_mpio_xfer_t xfer_mode)
```

Set the data transfer property list to specified transfer mode (`H5FD_MPIO_INDEPENDENT` or `H5FD_MPIO_COLLECTIVE`)

```
status = H5Dwrite(dset_id, H5T_NATIVE_INT, memspace, filespace, plist_id,
data)
```

```
herr_t H5Dwrite( hid_t dataset_id, hid_t mem_type_id, hid_t mem_space_id,
hid_t file_space_id, hid_t xfer_plist_id, const void * buf )
```

dataset_id: Identifier of the dataset to write to.

mem_type_id: Identifier of the memory datatype.

mem_space_id: Identifier of the memory dataspace.

file_space_id: Identifier of the dataspace in the file (in our case, slice of the file dataset)

xfer_plist_id: Identifier of a transfer property list for this I/O operation (in our case, specifies collective I/O)

buf: Start of buffer with data to be written to the file.

```
/*
* Close/release resources.
*/
H5Dclose(dset_id);
H5Sclose(filespace);
H5Sclose(memspace);
H5Pclose(plist_id);
H5Fclose(file_id);

MPI_Finalize();

return 0;
}
```

Data Analysis

Big Data – 5V's

Volume: Beyond what is handled by single machine or can be managed by conventional database. E.g., WWW

Velocity: How quickly data is updated (e.g., Internet traffic)

Variety: text, audio, video, sensor data...

Veracity: Data is noisy

Value: good data + good analysis generates value

Large scientific data

- Data generated by large experiments (e.g. LHC – 25 PB/year) – Volume, Velocity
- Data generated by large observations (e.g., DES – 2.5 TB/night; LSST 15 TB/neight) – Volume, Velocity, Veracity
- X-Informatics (bioinformatics, cheminformatics, environmental informatics...) – aggregation of many distinct information sources (publications, images, observations...) — Volume, Velocity, Variety, Veracity, Value

Characteristics

- Data (often) does not fit in memory – Need to repeatedly load and store data – Key performance issue is to minimize number of passes over data.
- CPU performance is less critical – interpreted languages such as Python are often used
- Frameworks are used to orchestrate data movement, and to hide underlying architecture
- Most often done in cloud systems – where each node has own disk (better if data is read multiple times)

Big Data + Big Compute

There is an increasing convergence of big data and HPC:

- Deep Learning (training neural networks)
- Analysis of simulation outputs
- Comparison of simulation to observation or experiment

Hadoop

Apache open source framework for storing and analyzing big data. Contains:

Hadoop Distributed File System (HDFS) – distributed file system built atop clusters of commodity machines

Hadoop MapReduce – an implementation of the MapReduce programming model.

Originally inspired by Google MapReduce and Google File System
Now used as part of larger packages, such as *Apache Spark*

Big Data Storage – HDFS

- Unstructured data – arrays a la HDF5 are not that useful
- Focus on streaming I/O – reading or writing large amounts of data, in no particular order (as distinct from random I/O)
- Relaxes POSIX semantics – no need for strict coherence
- Build atop large number of local file systems on commodity nodes
- Use replication to provide fault recovery
- Move computation to data, whenever possible